CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Elsevier - Publisher Connector

ELSEVIER

Journal of Computational and Applied Mathematics 55 (1994) 275–288

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

A recursive approach to local mesh refinement in two and three dimensions

Igor Kossaczký

Faculty of Electrical Engineering, Slovak Technical University, Ilkovičova 3, 812 19 Bratislava, Slovak Republic

Received 4 November 1992; revised 16 July 1993

Abstract

The newest vertex strategy for local refinement in two dimensions is reviewed and discussed. A recursive algorithm for local refinement of tetrahedral meshes in three dimensions based on similar bisection strategy is described. The recursive approach requires certain restrictions on the initial mesh. On the other hand, under this condition, the refinement process can be kept as local as needed, and it can be fully inverted. Simple data structures and derefinement algorithms are also outlined.

Keywords: Finite element method; Tetrahedral mesh; Adaptive refinement

1. Introduction

Local mesh modification is one of the most important aspects in the numerical solution of partial differential equations by means of adaptive finite element methods.

The crucial problem in the local mesh refinement is maintaining of the mesh conformity. One way to solve the problem is nonrecursive (see [1, 5, 6]): At first an element is divided, what in general breaks conformity, and then conformity is recovered dividing some other elements. Another recursive, approach (see [3, 4]) maintains conformity during the whole refinement process carefully controlling the order in which elements are divided. This approach requires certain preprocessing to provide for termination of the recursion. On the other hand, the refinement process can be kept as local as needed, and it can be fully inverted by a derefinement algorithm. Moreover, since nonconforming points never occur it is easy to implement.

At first we shortly explain the newest vertex strategy, discuss some aspects of the recursive approach of [3] in two dimensions and describe the corresponding derefinement algorithm. Then we introduce a similar three-dimensional refinement algorithm for tetrahedral meshes and give a bound on the length of recursion. We also outline the corresponding three-dimensional

0377-0427/94/$07.00 © 1994 Elsevier Science B.V. All rights reserved
SSDI 0377-0427(93)E0196-S

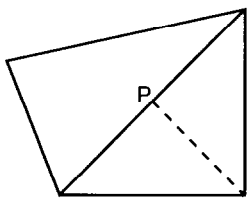

Fig. 1.

derefinement process and suitable data structure. The three-dimensional algorithm requires certain restrictions on the initial mesh. These conditions are discussed in the last part of the paper.

2. The two-dimensional case

2.1. Newest vertex bisection

The newest vertex bisection was introduced in [7] and it was also used in [1, 3, 4]. A triangle is divided to form two new *children* triangles by connecting one of vertices called a *peak* to the midpoint of the opposite edge called a *base* or a *refinement edge*. The new vertex created at a midpoint of the refinement edge is assigned to be the peak of children; hence the bisection process can proceed. It is shown in [7] that there are only four similarity classes of triangles created by this method.

2.2. Local refinement of a triangulation and conformity

A triangulation T is called *conforming* if for any two triangles $t, t' \in T$, $t \cap t'$ is empty, a common vertex or a common edge. Let T be a conforming triangulation. The bisection of a triangle $t \in T$ creates a *nonconforming point* P at the edge of a neighbor (see Fig. 1). Let each $t \in T$ have one refinement edge, and let $\Sigma \subseteq T$ be a set of those triangles of T which have to be divided. The following algorithm divides triangles of Σ and recovers conformity.

Algorithm 1.
while $\Sigma \neq \emptyset$ **do begin**
Bisect each $t \in \Sigma$;
Let now Σ be the set of those triangles with a nonconforming point;
end.

It was showed in [1] that the algorithm stops in a finite number of steps.

2.3. Recursive approach

In the approach introduced in [3, 4], nonconforming points never occur. This is achieved by dividing pairs of triangles that share a common edge rather than individual triangles. A triangle is

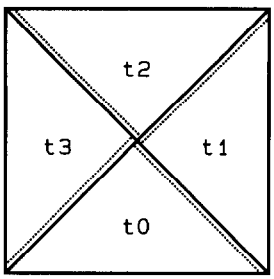

Fig. 2.

said to be *compatibly divisible* if its refinement edge is either the refinement edge of the triangle opposite its peak or a part of the boundary of the domain. If a triangle is compatibly divisible, then we divide the triangle and the neighbor opposite the peak simultaneously as a pair. If a triangle is not compatibly divisible, then after a single bisection of the neighbor opposite the peak, it will be. So in this case, we divide the neighbor by the same process first, and then divide the pair. This leads to the recursive algorithm.

Algorithm 2.
procedure divide_triangle(t)
if t is not compatibly divisible **then**
divide triangle(neighbor of t opposite the peak);
Divide by bisection the triangle t and the neighbor opposite the peak of t.

Refinement edges of the initial mesh cannot be selected freely. For some distribution of refinement edges the recursions do not terminate. Such initial mesh is illustrated in Fig. 2. The bisection of the triangle t_0 by divide_element(t_0) enforces the infinite sequence of recursive calls of divide_element with parameters $t_1, t_2, t_3, t_0, t_1, \ldots$.

The following theorem, proved in [3, 4], solves this problem. Let the triangles of the initial triangulation be assigned *generation* 1. Let children have generation $i + 1$ where i is the generation of the parent.

Theorem 1. *Let all triangles of the initial mesh be compatibly divisible. The length of the recursion in Algorithm 2 is bounded by the generation of triangle t.*

We will show another good and simple choice of refinement edges. Suppose that each triangle of the initial triangulation has the unique longest edge. If it is not true, then there exists a linear transformation of R^2 such that this assumption is met for the image of the initial triangulation. (Such transformation could be found in time proportional to the number of triangles.)

Theorem 2. *Let the refinement edge of each triangle of an initial triangulation T_0 be the longest edge of the triangle. Then Algorithm 2 stops on T_0 and also on every refinement of T_0, created by this method.*

Proof. Let T be a triangulation and refinement edges of triangles of T are defined. Let us consider a directed graph $\mathrm{RG}(T)$ called a *refinement graph of* T. The vertices of $\mathrm{RG}(T)$ are the triangles of T. Let $t, t' \in T$ be vertices of $\mathrm{RG}(T)$. There is a directed edge from t to t' in $\mathrm{RG}(T)$ if and only if t' is opposite the peak of t, but t is not opposite the peak of t'. It is easy to see that the sequence of recursive calls of divide_element, starting with a parameter $t \in T$, follows in $\mathrm{RG}(T)$ the path from t and stops at a leaf if it exists.

Note that the algorithm bisects only compatibly divisible triangles of T, i.e., sinks of $\mathrm{RG}(T)$. It can be also verified that the children triangles became sources of the refinement graph of the new refined triangulation. Hence, at the bisection step of the algorithm sinks are removed and new sources are added to $\mathrm{RG}(T)$. Thus if $\mathrm{RG}(T)$ is acyclic, then it also maintains this property during the refinement process.

Since the refinement edge of each triangle $t \in T_0$ is the longest edge of t, the refinement graph $\mathrm{RG}(T_0)$ is acyclic. Thus, the algorithm stops not only on the initial mesh T_0 but also on every refinement T created using this algorithm. □

The choice of the longest edge as the refinement edge appears to be good from a numerical point of view, but on the other hand, in this case the recursion could be very large. The refinement of an element can enforce the bisection of remote elements.

For purpose of a local refinement it could be more convenient to spend some preprocessing time and find such choice of refinement edges that all triangles are compatibly divisible, which implies that recursion will be as short as possible.

2.4. *Local derefinement of a triangulation*

Another useful property of the recursive approach is that it can be fully inverted. A pair of child triangles can be glued so that their parent is reconstructed. To achieve this we store with every triangle t the order in which vertices of t were created. This enables us to find out which neighbor is the *glue neighbor* of t — the other child of the parent of t (or a descendant of it). A triangle t is said to be *connectable*, if the newest vertex of t is also the newest vertex of the glue neigbor of t (i.e. the glue neighbor is the other child of the parent of t). Let v be the newest vertex of a triangle t, let t_1 be the glue neighbor of t, t_2 the second neighbor of t that shares v and t_3 the glue neighbor of t_2. It can be easily seen that if v is also the newest vertex of t_2 and both t and t_2 are connectable, then both t and t_2 can be glued without breaking conformity (see Fig. 3(c) and (d)). Otherwise, if v is not the newest vertex of t_2, then after gluing t_2 by the same process it will be (see Fig. 3(a) and (b)), and finally, if either t or t_2 is not connectable then after gluing t_1 or t_3 they will be (see Fig. 3(b) and (c)). Hence we have the recursive derefinement procedure.

Algorithm 3.
procedure glue_triangle(t)
Let v be the newest vertex of t;
Let t_2 be the neighbor containing v but not the glue neighbor of t;
if v is not the newest vertex of t_2 **then**
  glue_triangle(t_2);

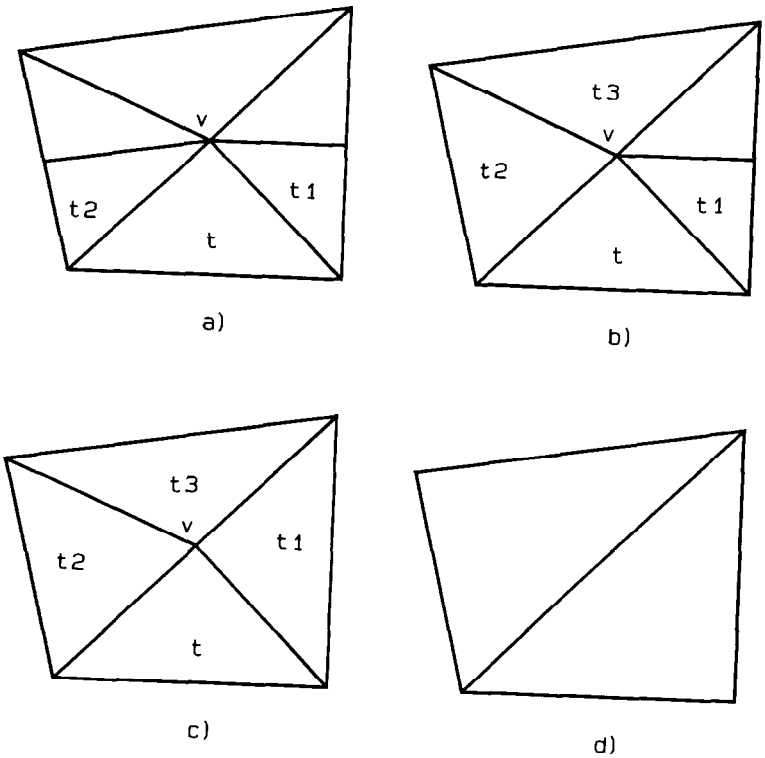

Fig. 3.

if t is not connectable **then**
  glue_triangle(glue neighbor of t);
Let t_2 be the neighbor containing v but not the glue neighbor of t;
if t_2 is not connectable **then**
  glue_triangle(glue neighbor of t_2);
Glue t and t_2 with their glue neighbors.

Suppose that triangulation is a refinement of an initial triangulation satisfying the compatibility condition in Theorem 1. It can be verified without difficulty that the recursive call in glue_triangle(t) occurs only if the argument has generation greater than the generation of t. Thus, the length of recursion is bounded by the highest generation.

3. The three-dimensional case

The newest vertex strategy was extended to three dimensions in [1]. For the local refinement of a tetrahedral mesh, there an algorithm identical to Algorithm 1 was proposed and it was proved that, under certain conditions on the initial position of the refinement edges, the process stops in a finite number of steps. These simple conditions can be easily fulfilled; for details see [1].

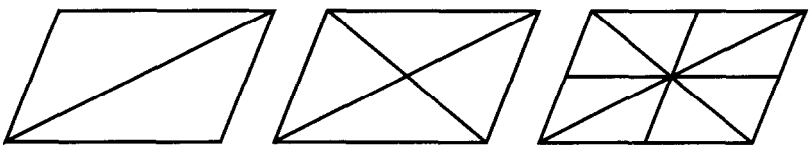

Fig. 4.

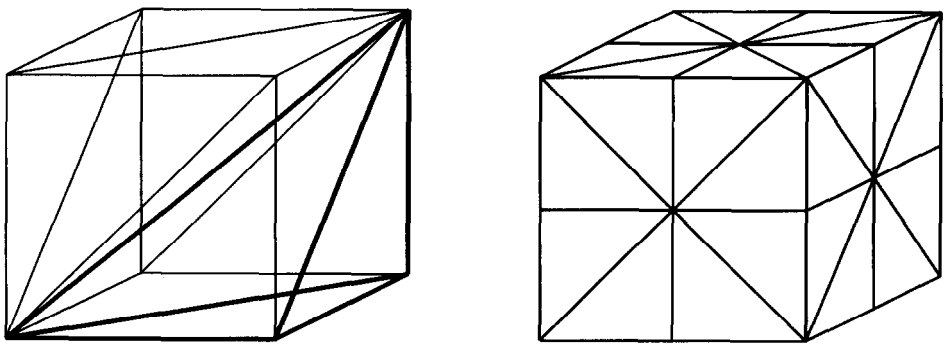

Fig. 5.

Unfortunately, the recursive version of this algorithm (see Algorithm 4 later) similar to Algorithm 2 does not work under such simple conditions. We do not know the conditions which can be fulfilled without difficulty for any initial tetrahedral mesh and under which a three-dimensional version of the statement of Theorem 2 holds. The aim of the rest of this paper is to extend the recursive approach of [3, 4] to three dimensions and prove a three-dimensional version of Theorem 1.

The idea of the recursive Algorithm 2 is that we always bisect pairs of adjacent triangles. This fact suggests another interpretation of the newest vertex strategy.

A triangle is divided as if it was a part of a parallelogram. The parallelogram is divided in two steps into four similar parallelograms, each consisting of two triangles (see Fig. 4).

At the first step the diagonal is the refinement edge of both triangles. At the second step the refinement edges are edges of the parallelogram.

3.1. Bisection of a tetrahedron

To divide a single tetrahedron t, it is supposed to be embedded into a parallelepiped M consisting of six tetrahedra as in Fig. 5(a). Three edges of each tetrahedron correspond to edges of M, two are diagonals of faces of M and one is the diagonal of M. The parallelepiped is divided by bisections in three steps into eight similar parallelepipeds consisting of six tetrahedra (see Fig. 5(b)). In one

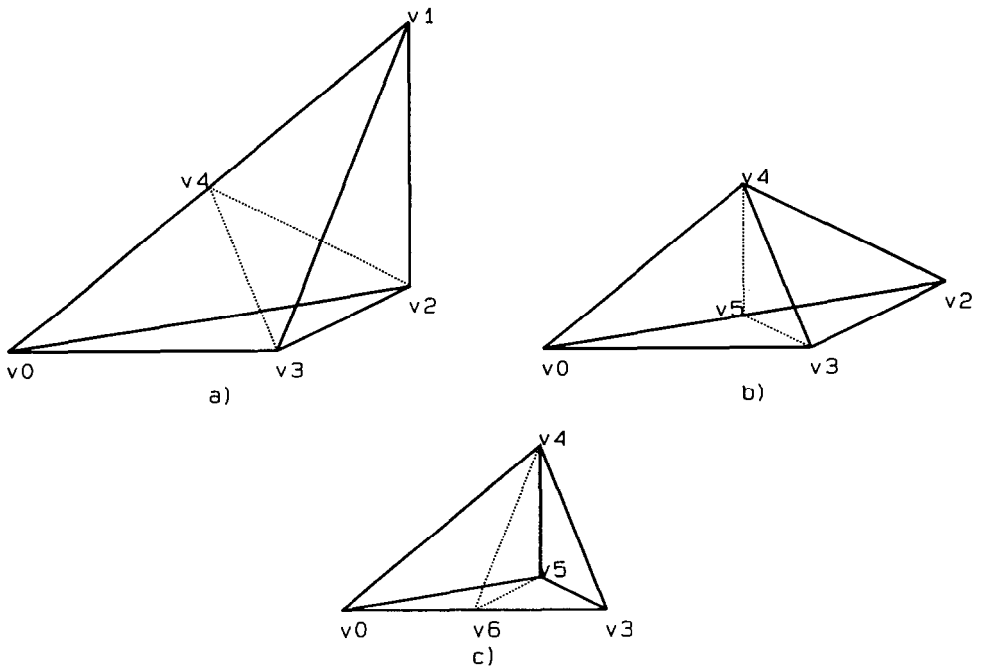

Fig. 6.

step each tetrahedron is bisected creating a new vertex in the middle of an edge, called the *refinement edge*, and connecting it to the opposite edge.

At the first step (Fig. 6(a)) the diagonal of M is the refinement edge. At the second step (Fig. 6(b)) refinement edges are diagonals of faces of M. At the third step (Fig. 6(c)) refinement edges are edges of M.

Now the refinement process can proceed again with step 1. It can be easily verified that there is only a finite number of similarity classes of tetrahedra created by this method. Hence, angles are bounded away from 0 to π. The refinement process on a tetrahedron, of course, need not start with step 1. We shall use the following notions.

Definition. Let t be a tetrahedron and P be a parallelepiped mentioned above. Let a mapping i_t be given from t to P, such that the image of t is one of the tetrahedra in steps 1, 2 or 3.

The mapping i_t will be called *initial embedding.*

The tetrahedron t is said to be of *type* 1, 2 or 3 according to the step.

Let e be an edge of t. e is said to have type "*edge*", "*face diagonal*" or "*diagonal*" if $i_t(e)$ is edge, face diagonal or diagonal of the parallelepiped.

An initial embedding of a tetrahedron t fully determines the refinement process on t. The notions *initial embedding*, *type of tetrahedron* and *type of edge* can be extended also to all child tetrahedra of t. (A child tetrahedron is embedded to P by the initial embedding of its ancestor, the type of an edge e is "edge", "face diagonal" or "diagonal" if $i_t(e)$ is parallel to an edge, a face diagonal or a diagonal of P.)

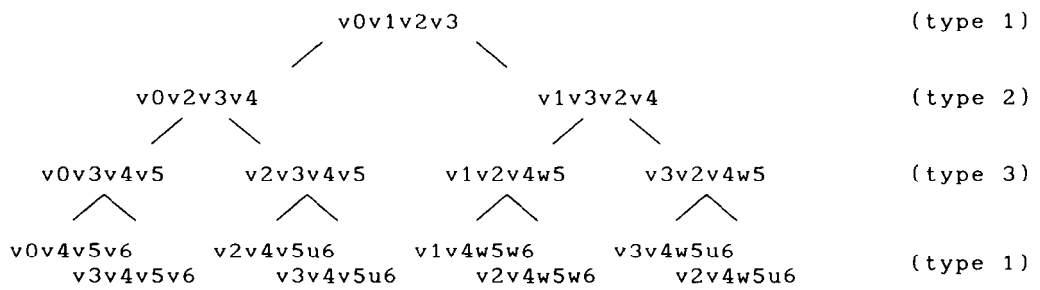

Fig. 7.

Though the process of dividing a tetrahedron seems to be rather complicated it can be easily implemented. With a tetrahedron t we store pointers (or indexes) to four vertices of t and the type of t. Pointers to the vertices are ordered so that the first two vertices represent the refinement edge of t and the fourth is the newest vertex of t. Let, for example, t be the tetrahedron of type 1 illustrated in Fig. 6(a). Then vertices will be ordered $v_0v_1v_2v_3$ and the order of vertices of tetrahedra created dividing t is illustrated in Fig. 7.

It is easy to see that the division process is fully determined by the order of vertices and the type. Indeed, the second (or the first, for the second child) vertex is simply removed and the new vertex is appended. The order of the remaining vertices is unchanged, except for the second child of elements with type 1, where a permutation occurs.

The method we have just described is identical to the method of [1]. Indeed, three steps correspond directly to the red–black–black cycle of [1]. We find the description of the process used here to be more suitable to describe conditions under which the recursive local refinement algorithm works.

3.2. Local refinement of a tetrahedral mesh

Let T be a tetrahedral mesh and for every $t \in T$ an initial embedding is given. We shall make the following assumptions.

(A1) For each pair of tetrahedra t, $t' \in T$, sharing common edge e, the type of e with respect to t and the type of e with respect to t' is the same.

(A2) All tetrahedra of T are of the same type.

Though for simple meshes containing a small number of tetrahedra it is not difficult to find initial embeddings satisfying assumption (A1) or both (A1) and (A2), we do not know if it is possible for arbitrary tetrahedral mesh. The methods how to solve this problem for some classes of initial meshes will be discussed in the last part of this paper.

The following statement follows immediately from the description of the refinement process.

Lemma 1. *Let t be a tetrahedron and let s be a face of t. If s does not contain the refinement edge of t, then it contains the refinement edge of a child of t.*

Let t be a tetrahedron and e be the refinement edge of t. Let s be a face of a tetrahedron t. If s contains the refinement edge of t, then also say that it is the refinement edge of the face s. Otherwise, the refinement edge of s is the refinement edge of a child of t. It can be verified without difficulty that the position of the refinement edge of a face depends only on the types of edges but not on the type of the tetrahedron. The following lemma is the immediate consequence of this fact.

Lemma 2. *Let the initial embeddings satisfying condition* (A1) *be given for a mesh* T. *Let* $t, t' \in T$ *be a pair of adjacent tetrahedra. If* t *contains the refinement edge of* t' *and vice versa, then they have the same refinement edge.*

The recursive two-dimensional algorithm is based on bisection of two triangles that share the same refinement edge. This idea can be extended to three dimensions in a natural way. We shall bisect all tetrahedra sharing a certain edge in one step. Let t be a tetrahedron and let e be the refinement edge of t. The tetrahedron t can be divided only if e is the refinement edge of each tetrahedron that contains e. We will say, like in the two-dimensional case that t is *compatibly divisible*. To make t compatibly divisible, the refinement procedure at first traverses tetrahedra around the refinement edge e. The process starts from t and when it reaches such tetrahedron d' that e is not its refinement edge, then, according to Lemmas 1 and 2, after division of d' by divide_element it will be such. This leads to the recursive Algorithm 4.

Let t be a tetrahedron. A tetrahedron t' containing the refinement edge of t is called a *refinement neighbor*. We can suppose that all edges of the mesh are directed. The first refinement neighbor of t to the left with respect to the direction of the refinement edge is called *left refinement neighbor*. Suppose, for simplicity that e is not a part of the boundary of the domain.

Algorithm 4.
Procedure refine_element(t)
Let e be the refinement edge of t.
{At first t is made compatibly divisible}
$d \leftarrow t$;
Let d' be the left refinement neighbor of d;
while $d' \neq t$ **do**
begin
if e is not the refinement edge of d' **then**
begin
divide_element(d');
$d' \leftarrow$ the left refinement neighbor of d;
end
$d \leftarrow d'$;
$d' \leftarrow$ the left refinement neighbor of d;
end
{now t is compatibly divisible}
Divide all tetrahedra sharing the refinement edge of t.

Similar to the two-dimensional case, we use the generation of an element to give a bound on the length of recursion. At first we prove a lemma that relates the generations of neighboring tetrahedra.

Lemma 3. *Let be given a mesh T_0 together with a system of initial embedding satisfying conditions* (A1) *and* (A2). *Let $t, t' \in T$ be a pair of neighboring tetrahedra and $s = t \cap t'$ be the common face. Then*

(i) *if s contains the refinement edge of both t and t', then*

$$\text{generation}(t) = \text{generation}(t');$$

(ii) *if s contains the refinement edge of t but does not contain the refinement edge of t', then*

$$\text{generation}(t') = \text{generation}(t) - 1;$$

(iii) *if s does not contain the refinement edge of any of t and t', then*

$$\text{generation}(t) = \text{generation}(t').$$

Proof. We prove this by induction. To prove the conclusion for T_0, it is sufficient to show that assumptions of the case (ii) cannot be fulfilled. Suppose that the refinement edge of t is not the refinement edge of t'. Let d be the child of t', which is a neighbor of t. It follows by Lemmas 1 and 2 that t and d have now the same refinement edge. Hence t', t and d have the same type. But it is impossible since the types of an element and its children are different.

Suppose the conclusion holds for T, a refinement of T_0, and consider the mesh T' obtained by dividing a compatibly divisible element of T. Let t, t' be elements of T'. If both t and t' are also elements of T, then the conclusion holds by the induction hypothesis. If neither t nor t' belongs to T, then they have been just created, and so their parents had the same refinement edge; hence they must be of the same type. It can be verified in the same way as for the initial mesh T_0 that case (ii) is unreachable; thus the conclusion holds. Finally, suppose that $t \in T$, $t' \notin T$, and $d \in T$ is the parent of t'. Obviously, the face $s = t \cap d$ does not contain the refinement edge of d, but according to Lemma 2, it contains the refinement edge of t'. If s contains also the refinement edge of t, then, by the induction hypothesis,

$$\text{generation}(t) = \text{generation}(d) + 1 = \text{generation}(t').$$

Hence, the conclusion is valid. If s does not contain the refinement edge of t, then by the induction hypothesis,

$$\text{generation}(t) = \text{generation}(d) = \text{generation}(t') - 1;$$

so again the conclusion holds. □

The following theorem is an immediate consequence of Lemma 3(ii).

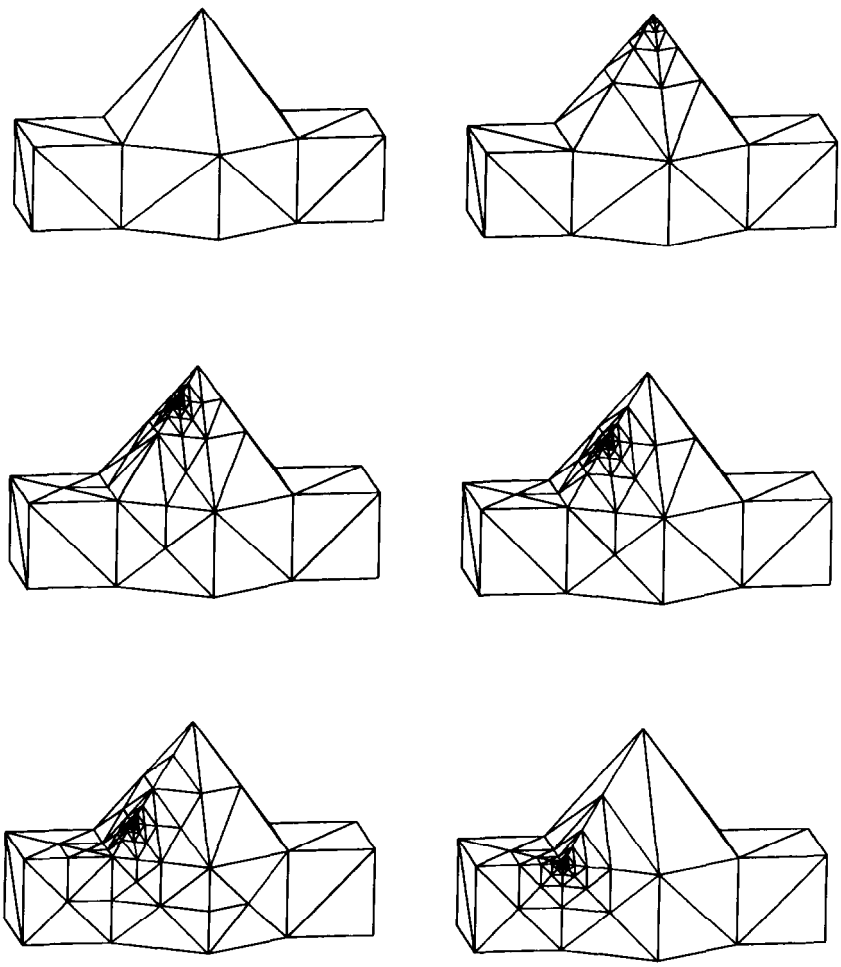

Fig. 8.

Theorem 3. *Let a mesh T_0 be given together with a system of initial embedding satisfying conditions* (A1) *and* (A2). *Then the length of the recursion in the procedure* divide_element(t) *is bounded by the generation of triangle t.*

3.3. Derefinement

The refinement process can also be easily inverted. Once we know the data of children of an element t, it can be reconstructed. Because of the problem with the second child in the first step, it is necessary to know which child is the first and which the second. It can be easily achieved indexing elements so that the first child inherits the index of the parent and the second child is given a new larger index. A mesh modified by a sequence of local refinement and derefinement instructions is illustrated in Fig. 8.

4. Initialization

The rest of the paper will be devoted to the problem of initial embeddings. As it was stated, we do not know if there exists a system of initial embeddings satisfying conditions (A1) and (A2) for an arbitrary mesh. One simple way how to avoid this problem, in the general case, is to look for initial embeddings for a refinement of the initial mesh rather than for the initial mesh itself.

Let T be an initial mesh. We divide each tetrahedron $t \in T$ into twelve tetrahedra in two steps:

(1) A tetrahedron t is divided into four child tetrahedra inserting a new vertex into the center of gravity of t and connecting it to vertices of tetrahedron t.

(2) Each child tetrahedra is divided into three new tetrahedra inserting a new vertex in the center of gravity of the face of the original tetrahedron and connecting it to vertices of the child tetrahedron.

For the resulting twelve tetrahedra, initial embeddings are defined. They are mapped so that edges of the original tetrahedron t are of type "face diagonal", edges added in the first step are of type "diagonal" and edges added in the second step are of type "edge". Hence, all tetrahedra are of type 2.

These two division steps could be incorporated into the local refinement algorithm without difficulty as the first two nonstandard steps. A shortcoming of this method is that at this first two steps, angles which are unnecessarily small are created and, on the other hand, the diameters of tetrahedra are not sufficiently reduced.

Another and a probably more convenient way how to solve the problem of initial embeddings is to modify the existing mesh generators so that they generate a mesh together with initial embeddings satisfying the required conditions. Program packages for the finite element method contain various mesh generators based on different meshing strategies. For example, the package MODULEF [2] contains a generator that creates a hexahedral mesh, which can be subsequently refined to a tetrahedral mesh. A generator is also implemented for cylindrical domains that creates a pentahedral mesh from a triangulation of the base of a cylindrical domain. We shall outline how these generators could be modified.

Let H be hexahedral mesh. We shall divide each hexahedron $h \in H$ into a tetrahedron so that faces of h will be bisected into two triangles and we shall define initial embeddings for this tetrahedron so that edges of h will receive type "edge" and diagonals of faces of h will be of type "face diagonal".

Since a hexahedron can be continuously mapped on a parallelepiped, every hexahedron $h \in H$ can be simply divided into six tetraheda of type 1 as in Fig. 3(a). An algorithm that converts a hexahedral mesh into tetrahedral should only determine how to map each hexahedron h into a parallelepiped. Let $h, h' \in H$ be two hexahedra sharing a common face f. They must be divided so that the face f is bisected in the same way, otherwise a nonconforming mesh is generated. Unfortunately, if we admit that faces of hexahedra do not have to be plane, then there exists a hexahedral mesh for which this requirement cannot be fulfilled. To avoid this difficulty we shall divide hexahedra into six tetrahedra of type 1 or into twelve tetrahedra of type 2.

Suppose that an algorithm divides hexahedra one by one into tetrahedra of type 1 as long as it is possible. When this process reaches the situation where neighbors of a hexahedron h are already divided and faces of h are bisected so that h cannot be divided into tetrahedra of type 1, then h is divided into tetrahedra of type 2. A new vertex v is inserted into the center of gravity of h and,

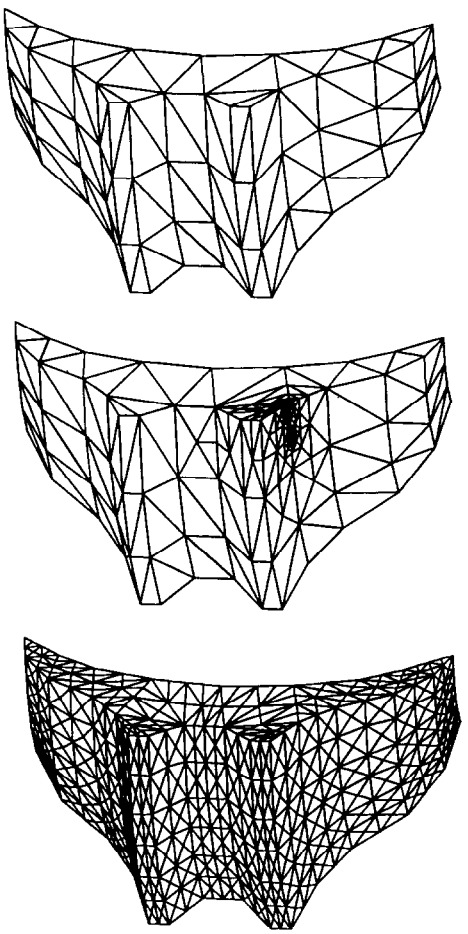

Fig. 9.

connecting v with vertices of h, h is divided into twelve tetrahedra. It is easy to see that we can define initial embeddings for these tetrahedra so that edges of h will have type "edge", diagonals of faces of h will be of type "face diagonal" and tetrahedra will be of type 2.

A tetrahedral mesh T created by this method satisfies the property (A1). Though property (A2) is not fulfilled, it could be proved that the length of the recursion in procedure divide_element(t) (see Algorithm 4) for a tetrahedron $t \in T$ is bounded by generation(t) + 1. The proof is similar to that of Lemma 3.

Now we shall consider a generator for cylindrical domains. Usually, such a generator creates a triangulation T_0 of the base of a cylindrical domain. Then this triangulation is extended to a pentahedral mesh P, which can be subsequently refined into a tetrahedral mesh T.

Let T_0 be a triangulation of the base Ω_0 of a domain Ω. At first we select for every triangle $t \in T_0$ a refinement edge so that all triangles of the triangulation T_0 are compatibly divisible. Let us suppose that each refinement edge is a common refinement edge of a pair of neighboring triangles. (If an edge e is a refinement edge of only one triangle then it must be a part of the boundary of T_0.

In this case, we, hypothetically, add a new triangle containing the edge e to the triangulation to create a pair.)

Every pair of compatibly divisible triangles represents the base of a hexahedron consisting of two neighboring pentahedra of P. The system of hexahedra is refined into a tetrahedral mesh T and initial embeddings are defined as it was described above. A hexahedron h consisting of two pentahedra $p, p' \in P$, can be divided into six (or twelve) tetrahedra so that both p and p' are divided into three (or six) tetrahedra. Thus, T is a refinement of the pentahedral mesh P. A mesh generated by this method together with a global and local refinement of it is illustrated in Fig. 9.

References

[1] E. Bänsch, An adaptive finite-element strategy for the three-dimensional time-dependent Navier–Stokes equations, *J. Comput. Appl. Math.* **36** (1991) 3–28.

[2] P.L. George, MODULEF, User guide No. 3, INRIA, 1992.

[3] W.F. Mitchell, Unified multilevel adaptive finite element methods for elliptic problems, Ph.D. Thesis, Report no. UIUCDCS-R-88-1436, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1988.

[4] W.F. Mitchell, Adaptive refinement for arbitrary finite-element spaces with hierarchical bases, *J. Comput. Appl. Math.* **36** (1991) 65–78.

[5] M.-C. Rivara, Selective refinement/derefinement algorithms for sequences of nested triangulations, *Internat. J. Numer. Methods Engrg.* **28** (1989) 2889–2906.

[6] M.-C. Rivara, Local modification of meshes for adaptive and/or multigrid finite-element methods, *J. Comput. Appl. Math.* **36** (1991) 79–89.

[7] E.G. Sewell, Automatic generation of triangulation for piecewise polynomial approximation, Ph.D. Thesis, Purdue Univ., West Lafayette, IN, 1972.